REMARKS:

Claims 1-55 were presented for examination and were pending in this application. In an Official Action dated March 17, 2008, claims 1-55 were rejected. Applicants thank Examiner for examination of the claims pending in this application and address Examiner's comments below.

Applicants herein amend claims 17, 19 and 53. Applicants have added new claims 56 and 57 herein. These changes are believed not to introduce new matter, and their entry is respectfully requested. The claims have been amended to expedite the prosecution of the application in a manner consistent with the Patent Office Business Goals, 65 Fed. Reg. 54603 (Sept. 8, 2000). In making these amendments, Applicants have not and do not narrow the scope of the protection to which Applicants consider the claimed invention to be entitled and do not concede that the subject matter of such claims was in fact disclosed or taught by the cited prior art. Rather, Applicants reserve the right to pursue such protection at a later point in time and merely seek to pursue protection for the subject matter presented in this submission.

Based on the above Amendment and the following Remarks, Applicants respectfully request that Examiner reconsider all outstanding objections and rejections, and withdraw them.

Objection to the Specification

The Examiner has objected to the specification because of informalities. Applicants have amended the specification accordingly. Thus, Applicants respectfully submit that all informalities with respect to the specification have been resolved.

Objections to the Claims

The Examiner has objected to claim 53 because of informalities. Applicants have amended claim 53 accordingly. Thus, Applicants respectfully submit that all informalities with respect to claim 53 have been resolved.

In view of these amendments, Applicants respectfully request that Examiner withdraw the objections to claim 53.

Response to Rejection Under 35 USC § 103(a) to Gee

The Examiner rejects claims 1, 29-33 and 42-47 under 35 USC §103(a) as allegedly being unpatentable over U.S. Patent No. 6,374,286 to Gee et al. ("Gee"). This rejection is respectfully traversed.

Claim 1 recites:

> a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

The hardware thread scheduler recited in claim 1 is configurable so the processor switches from one thread to another thread according to a predetermined fixed schedule. This is greatly beneficial as it provides a predictable execution time for threads, allowing each thread to be executed for a predetermined number of instruction cycles specified by the predetermined fixed schedule.

In contrast, Gee discloses using a single Java embedded microprocessor (JEM) to execute multiple Java Virtual Machines (JVMs). Each JVM is assigned a fixed area of memory and allotted a fixed amount of time in which to operate. After the fixed amount of time, the JEM invokes a master JVM to handle system duties and uses the master JVM to subsequently invoke another JVM. Gee, col. 3, lines 50-65. To switch between virtual machines, the JEM executes thread control blocks (TCBs) and executive control blocks (ECBs), which are both constructed to look like JAVA objects to simplify manipulation by JAVA software. Gee, col. 19, lines 40-59. Hence, the JVM does not use a hardware thread scheduler, but uses software processes (the TCBs and ECBs) to perform context switching between virtual machines. As software, the ECBs of Gee must be executed by the JEM in order to specify a thread switching schedule. Because of this, the ECBs and TCBs of Gee execute at too high a level to allocate specific numbers of instruction cycles, as claimed.

Gee discloses that execution of the ECB causes "interstitial" activity while switching between threads, which consumes a number of cycles, as illustrated in FIG. 14. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. This "interstitial" activity performs various system-level functions for the JEM, so that the activity does not perform for a defined number of cycles, causing imprecise context switching which renders Gees incompatible with cycle-specific scheduling. For example, an ECB indicating that an application should be executed every 100 time units, for example, could not indicate a specific number of cycles, because any varying number of cycles could be lost while performing the "interstitial" activity necessary to change contexts and subsequently switching to a new thread.

Gee discloses that an ECB uses a "piano roll" scheduler to process periodic threads using a list of entries which are periodically relayed in a fixed order responsive to receiving

an interrupt signal. Gee, col. 20, lines 44-53. While the "piano roll" schedule describes an execution sequence, the schedule merely approximates when a thread is initiated, and does not account for the overhead involved with performing a context switch to the master JVM to perform system functions then performing a second context switch to the desired thread necessary to switch to a different thread. When switching between JVMs, Gee requires performance of interstitial activity after completion of a first JVM and prior to beginning to process a second JVM. When switching between JVMs, an initial context switch occurs from the executing JVM to an executive, or master JVM, which performs housekeeping operations, service interrupts and starts a "proxy thread" to prepare the next thread for execution by setting appropriate flags and checking appropriate interrupts. The proxy thread then pushes the flags and subsequent JVM onto a stack for execution. Gee, col. 23, lines 30-43. While the scheduling in Gee identifies when a thread switch occurs, it does not specify the number of cycles during which a thread is executed. The interstitial activity required by Gee introduces overhead into thread switching so that a portion of the time allocated for thread execution is used for system functions and to re-load an execution stack with the next thread. Thus, the piano roll schedule does not allocate a specific number of cycles for different threads, but merely identifies when a thread switch, and the associated overhead needed to complete the previous thread and load the new thread for execution, is scheduled to begin.

Executing the master JVM between threads causes gaps between execution of threads where system functions, such as interrupt handling, are performed and interrupts applicable to the next thread are identified. Gee, col. 23, lines 30-43; col. 23, line 65 to col. 24, line 13. Thus, the "piano roll" schedule causes context switching from a first thread to a master JVM,

which performs system housekeeping functions before switching to a second thread. The "interstitial" activity performed by the master JVM to save user of state information, handle interrupts and perform other system functions causes certain cycles to be used for the overhead of context switching rather than thread execution. Gee, FIG. 14; col. 25, line 24 to col. 26, line 7.

Hence, the "piano roll" scheduler does not cause "thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed. Therefore, Gee does not disclose the "predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 1.

Similarly, claim 46 recites:

> switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles

As discussed above, the TCBs and ECBs disclosed in Gee are software processes which must be executed by the JEM to determine thread switches. Although the ECBs include "piano roll" schedulers describing thread selection order, context switches between

threads cause "interstitial" activity when switching between threads, which consumes a number of cycles, as depicted in FIG. 14 of Gee. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. As this "interstitial" activity performs various system functions for the JEM, there is an overhead associated with switching between contexts introduced by the "interstitial" activity which causes imprecise scheduling of switching between threads, rendering Gee incompatible with cycle-specific scheduling, as claimed. For example, an ECB indicating that an application should be executed every 100 time units, for example, could not indicate a specific number of cycles, because any varying number of cycles could be lost performing the "interstitial" activity and subsequently switching to a new thread. Hence, the ECB of Gee assigns processing time and executes at too high a level to allocate specific numbers of instruction cycles, contrary to the claimed invention.

Thus, Gee fails to disclose that "the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 46.

As claims 29-32 and 42-45 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 29-32, 42-45.

As claim 47 is dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claim 47.

Applicants respectfully submit that for at least these reasons claims 1, 29-32, 42-47 are patentably distinguishable over the cited reference. Thus, Applicants kindly request withdrawal of these rejections.

Response to Rejection Under 35 USC § 102(e) in to Joy

The Examiner rejects claims 2-4, 13, 16-17 and 19-24 under 35 USC § 102(e) as allegedly being anticipated by U.S. Patent No. 6,542,991 to Joy et al. ("Joy"). This rejection is respectfully traversed.

Claim 17 recites:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule by controlling whether the execution pipeline retrieves instructions from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles without incurring a time penalty in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes. (emphasis added)

Similarly, claim 19 recites:

> switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector. (emphasis added)

Switching threads between consecutive instruction cycles beneficially allows switching between one program context and another without incurring any time penalty. Switching from one thread to another between the end of an execution cycle before the beginning of a next consecutive instruction cycle beneficially allows switching to occur without the loss of any execution cycles. To switch between the first and second threads

without incurring any time penalty, the claimed invention modifies which set of storage devices provides instructions to the execution pipeline.

In contrast, Joy simply describes conventional event driven thread scheduling in a multithreaded processor. The conventional multithreading described in Joy is intended for utilizing processor resources efficiently when a thread is stalled. See Joy, col. 6, lines 21-24. That is, the multithreading in Joy is for efficient use of the processor, to reduce "wasted cycle times resulting from stalling and idling." See Joy, col. 2, lines 17-19. The thread switch logic of Joy includes "multithreaded-type functionality in response to an exception condition." Joy, col. 15, lines 8-11. Context switches in Joy "typically are made in response to interrupts, including hardware and software interrupts, both internal and external, of a processor." Joy, col. 14, lines 62-63. While Joy's system includes "fast thread-switching," where timing of the thread switching process is described in detail, there is no mention of "consecutive cycles" or "zero-time" switching, as claimed. Joy, col. 3, lines 28-56.

The fastest thread switching described in Joy requires at least one overhead cycle. Joy, col. 16, lines 1-5. As part of this overhead, Joy discloses that "thread switch logic generates the TID signal with a thread switch delay or overhead of one processor cycle." Thus, Joy discloses a very small delay in switching, but nonetheless Joy discloses a delay associated with responding to the generated TID signal. Even the most ambitious portions of Joy teach the need of an overhead of at least one processor cycle. This overhead is caused by Joy's need to, responsive to the TID, save and restore the processor state associated with the currently executed thread and the thread to be executed. Joy, col. 15, lines 1-7. To perform the context switch responsive to the TID signal, Joy requires that states from the first thread are saved and assigning new states to the second thread. Joy, col. 6, lines 42-48. To save

and restore a processor state, a store operation and a load operation are executed to transfer the current state to a memory and load data from a memory to the processor. While Joy provides that the thread switch logic provides "fast, nanoseconds range, context switching," the context switching in Joy executes instructions to save and restore processor state. Joy, col. 15, lines 55-58.

In contrast, the claimed invention modifies whether the execution pipeline receives instructions from the first or second set of data storage devices. Rather than switch instructions responsive to a generated signal, the claimed invention directly fetches instructions from different storage devices responsive to the execution scheduler. For example, page 26, line 15 through page 27, line 2 of the specification and Figure 8 describe one implementation of the claimed invention. As described in the specification, multiple storage devices, shown in Figure 8 as "Flash A Fetch," "Flash B Fetch," "Flash C Fetch," "Flash D Fetch" and "Shadow SRAM," provide instructions to an execution pipeline based on the output from a hardware thread scheduler, illustrated in Figure 8 as "Post Fetch Select." Hence, the execution pipeline is capable of executing instructions from different threads during each instruction cycle by merely accessing a different storage device in each instruction cycle. For example, a context number in an instruction specifies which data storage device provides instructions to the execution pipeline, allowing the execution pipeline to directly retrieve the instructions from the identified data storage device. An example of this zero-time context switching is described in the specification at page 18, line 1 to page 19, line 5. Hence, the claimed invention allows retrieval and execution of instructions from different threads by modifying the register from which an execution pipeline retrieves instructions, allowing each instruction cycle to execute instructions from

different threads, while Joy requires that threads be loaded into the processor for execution, introducing overhead into thread switching caused by the time necessary for loading and storing processor state information. Hence, the claimed invention uses the content of the instruction to determine from which data storage device to retrieve instructions.

Hence, Joy does not switch threads between consecutive instruction cycles, but performs a context switching operation where processor states are saved and restored, introducing overhead into the thread switch. Joy discloses accelerating conventional thread switching to provide context switching in "nanosecond ranges." Joy, col. 7, lines 50-52. By "controlling whether the execution pipeline retrieves instructions from the first set of data storage devices or the second set of data storage devices," the claimed invention allows for execution of different threads merely during each instruction cycle by merely specifying what set of data storage devices are used to provide instructions to the execution pipeline. Hence, unlike the claimed invention, Joy is incompatible with switching without incurring a time penalty, as Joy explicitly discloses that switching *between consecutive instruction cycles* in response to the hardware thread scheduler is not possible.

Thus, the claimed "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule by controlling whether the execution pipeline retrieves instructions from the first set of data storage devices or the second set of data storage devices" and "thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles without incurring a time

penalty in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes" is not disclosed by the cited references, both alone and in combination. Therefore, Applicants submit that claims 17 and 19 are patentable over the cited art and request that the rejection be withdrawn.

As claims 2-4, 13 and 16 are dependent from claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 2-4, 13 and 16.

As claims 20-24 are dependent from claim 19, all arguments advanced above with respect to claim 19 are hereby incorporated so as to apply to claims 20-24.

Accordingly, for at least the reasons set forth above, claims 2-4, 13, 16-17 and 19-24 are patentable over the cited reference.

Response to Rejection Under 35 USC 103(a) in View of Gee and Borkenhagen

The Examiner rejects claims 1, 29-33, 42-43 and 45-47 under 25 USC § 103(a) as allegedly being unpatentable over U.S. Patent No. 6,076,157 to Borkenhagen et al. ("Borkenhagen") in view of Gee. This rejection is respectfully traversed.

Claim 1 recites:

> a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

The hardware thread scheduler recited in claim 1 is configurable so the processor switches from one thread to another thread according to a predetermined fixed schedule. This is greatly beneficial as it provides a predictable execution time for threads, allowing

each thread to be executed for a predetermined number of instruction cycles as specified in the predetermined fixed schedule.

In contrast, Borkenhagen discloses a system and method for data processing in a multithreaded processor where "[t]he thread switch logic has a time-out register which forces a thread switch when execution of the active thread in the multi-threaded processor exceeds a programmable period of time." Borkenhagen, Abstract. The time-out register in Borkenhagen is used "to force a thread switch to the dormant thread after some time if no useful processing is being accomplished to prevent the system from hanging." Borkenhagen, col. 14, lines 49-51. This forced thread switch prevents a thread from "spinning in a loop unable to do useful work" because it is unable to acquire ownership of, or access to, a necessary resource. Borkenhagen, col. 14, lines 31-34. Thus, the time-out register in Borkenhagen does not specify the processing time allocated to the first and second threads, but rather specifies a maximum time the first and second threads can be inactive before forcing a thread switch.

As the Examiner admits, Borkenhagen does not disclose "a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles." However, as discussed above, Gee does not remedy the deficient disclosure of Borkenhagen.

As discussed above, the TCBs and ECBs disclosed in Gee are software processes which must be executed by the JEM to determine thread switches. Although the ECBs include "piano roll" schedulers describing thread selection order, context switches between threads cause "interstitial" activity when switching between threads, which consumes a number of cycles, as depicted in FIG. 14 of Gee. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. As this "interstitial" activity performs various system functions for the JEM, there is an overhead associated with switching between contexts introduced by the "interstitial" activity which causes imprecise scheduling of switching between threads, rendering Gee incompatible with cycle-specific scheduling, as claimed. For example, an ECB indicating that an application should be executed every 100 time units, for example, could not indicate a specific number of cycles, because any varying number of cycles could be lost performing the "interstitial" activity and subsequently switching to a new thread. Hence, the ECB of Gee assigns processing time and executes at too high a level to allocate specific numbers of instruction cycles, contrary to the claimed invention.

Similarly, claim 46 recites:

> switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles

As discussed above, Borkenhagen discloses using a time-out register to force a thread switch when the currently executing thread is inactive or fails to produce a useful result for a defined number of cycles to prevent a background thread from spinning in a loop. Thus, the

time-out register in Borkenhagen specifies a maximum number of cycles in which a thread can be inactive or fail to produce useful output before forcing a thread switch. The time-out register in Borkenhagen allows a thread to continue executing indefinitely and does not switch threads until the executing thread fails to produce useful output or becomes inactive for a defined number of cycles. Thus, Borkenhagen fails to disclose that "the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 46.

Gee does not remedy this deficient disclosure, as discussed above, but discloses using software processes executed by the JEM to determine thread switches. Although these processes use "piano roll" schedulers to describe thread selection order, context switches between threads cause "interstitial" activity when switching between threads, which consumes a number of cycles, as depicted in FIG. 14 of Gee. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. As this "interstitial" activity performs various system functions for the JEM, it introduces an overhead with switching between contexts introduced by the "interstitial" activity, causing imprecise scheduling of switching between threads and rendering Gee incompatible with cycle-specific scheduling, as claimed. For example, an software process indicating that an application should be executed every 100 time units, for example, could not indicate a specific number of cycles, because any varying number of cycles could be lost performing the "interstitial" activity and subsequently switching to a new thread. Hence, the software processes of Gee assign processing time and executes at too high a level to allocate specific numbers of instruction cycles, contrary to the claimed invention.

As claims 29-32 and 42, 43 and 45 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 29-32, 42, 43 and 45.

As claim 47 is dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claim 47.

Applicants respectfully submit that for at least these reasons claims 1, 29-32, 42, 43 and 45-47 are patentably distinguishable over the cited reference. Thus, Applicants kindly request withdrawal of these rejections.

Response to Rejection Under 35 USC 103(a) in View of Joy and Ramakrishnan

The Examiner rejects claims 5-12, 18 and 25-28 under 25 USC § 103(a) as allegedly being unpatentable over Joy in view of U.S. Patent No. 6,085,215 to Ramakrishnan et al. ("Ramakrishnan"). This rejection is respectfully traversed.

As claims 5-12 and 18 are dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 5-12 and 18.

Ramakrishnan is cited to make up for Joy's lack of "thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The switching in Ramakrishnan, like in Joy, is conventional context switching involving "an associated overhead in invoking the new thread." (Ramakrishnan, col. 12, lines 61-62, see also, col. 13, lines 6-7 "avoid time consuming context switching"). Ramakrishnan does not remedy the deficiencies of Joy discussed above. Hence, the combination of Joy and

Ramakrishnan still fails to disclose the "between consecutive instruction cycles" switching recited in claim 17.

Accordingly, for at least the reasons set forth above, claims 5-12, 18 and 25-28 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

Response to Rejection Under 35 USC 103(a) in View of Gee and Ramakrishnan

The Examiner rejects claims 34-41 and 48-55 as allegedly being unpatentable over Gee in view of Ramakrishnan. This rejection is respectfully traversed.

As claims 34-41 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 34-41. As claims 48-55 are dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan is cited to make up for Gee's lack of "a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The scheduler in Ramakrishnan also fails to disclose "a predetermined fixed schedule" for allocating thread processing time. Hence, Ramakrishnan does not remedy the deficiencies of Gee.

Accordingly, for at least the reasons set forth above, claims 34-41 and 48-55 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

Response to Rejection Under 35 USC 103(a) in View of Joy and Borkenhagen

The Examiner rejects claim 14 as allegedly being unpatentable over Joy in view of Borkenhagen. This rejection is respectfully traversed.

As claim 14 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 14.

Borkenhagen is cited to make up for the deficiency of "storing said second thread state of said processor during execution of said first program thread" in Joy. However, the combination of Joy and Borkenhagen still fails to teach or suggest the "between consecutive instruction cycles" switching recited in claim 17. The system disclosed in Borkenhagen also incurs the conventional "latency and performance penalties associated with switching threads." Borkenhagen, col. 15, lines 37-38. Borkenhagen explicitly discloses:

> In the multithreaded processor in the preferred embodiment described herein, this latency includes the time required to complete execution of the current thread to a point where it can be interrupted and correctly restarted when it is next invoked, the time required to switch the thread-specific hardware facilities from the current thread's state to the new thread's state, and the time required to restart the new thread and begin its execution.

Borkenhagen col. 15, lines 38-46. Thus, Borkenhagen also fails to disclose "the pipelined processor switches from said first state to said second thread state between consecutive instruction cycles," so Borkenhagen fails to cure the deficiencies of Joy.

Accordingly, for at least the reasons set forth above, claim 14 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

Response to Rejection Under 35 USC 103(a) in View of Joy and Levy

The Examiner rejects claim 15 as allegedly being unpatentable over Joy in view of U.S. Patent No. 6,314,511 to Levy et al. ("Levy"). This rejection is respectfully traversed.

As claim 15 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 44.

Levy is cited to make up for Joy's lack of "said first set of data storage devices comprises registers shared by a plurality of threads." However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of the time necessary for switching between threads, but merely discloses a configuration for a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. As Levy does not disclose "the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles," it fails to remedy the deficient disclosure of Joy.

Accordingly, for at least the reasons set forth above, claim 15 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

Response to Rejection Under 35 USC 103(a) in View of Borkenhagen and Gee in Further View of Ramakrishnan

The Examiner rejects claims 34-41 and 48-55 as allegedly being unpatentable over Borkenhagen in view of Gee in further view of Ramakrishnan. This rejection is respectfully traversed.

As claims 34-41 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 34-41. As claims 48-55 are dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan is cited to make up for Borkenhagen and Gee's lack of "a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The scheduler in Ramakrishnan also fails to disclose "a predetermined fixed schedule" for allocating thread processing time. Hence, Ramakrishnan does not remedy the deficiencies of the combination of Borkenhagen and Gee.

Accordingly, for at least the reasons set forth above, claims 34-41 and 48-55 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

Response to Rejection Under 35 USC 103(a) in View of Borkenhagen in View of Gee and Levy

The Examiner rejects claim 44 as allegedly being unpatentable over Borkenhagen in view of Gee and in further view of Levy. This rejection is respectfully traversed.

As claim 44 is dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claim 44.

Levy is cited to make up for the combination of Gee and Borkenhagen's lack of "said first set of data storage devices comprises registers shared by a plurality of threads." However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of switching threads according to "a predetermined fixed schedule," but merely discloses a configuration for a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. As Levy does not disclose "causing thread-switching at a fixed time according to a predetermined fixed schedule," it fails to remedy the deficient disclosure of the combination of Borkengagen and Gee.

Accordingly, for at least the reasons set forth above, claim 44 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

New claims 56 and 57 have been added. Support for claims 56 and 57 are found throughout the specification, for example at page 18, lines 1-7. New claim 56 depends from claim 19 as well as recites additional patentable features, such as "communicating the context number associated with the second program thread to the execution pipeline," and "loading

instructions from the second set of data storage devices into the execution pipeline." New claim 57 depends from claim 17 and also recites additional patentable features, such as "communicating the context number associated with the second program thread to the execution pipeline," and "loading instructions from the second set of data storage devices into the execution pipeline."

Conclusion

In sum, Applicants respectfully submit that claims 1-57, as presented herein, are patentably distinguishable over the cited references. Therefore, Applicants request reconsideration of the basis for the rejections to these claims and request allowance of them.

In addition, Applicants respectfully invite Examiner to contact Applicants' representative at the number provided below if Examiner believes it will help expedite furtherance of this application.

Respectfully Submitted,
NICHOLAS J. KELSEY, ET AL.

Date: September 17, 2008

By: /Brian G. Brannon/
Brian G. Brannon, Registration No. 57,219
FENWICK & WEST LLP
801 California Street
Mountain View, CA 94041
Phone: (650) 335-7610
Fax: (650) 938-5200
E-Mail: bbrannon@fenwick.com

20880/05093/DOCS/1936635.1